An Online Payment  
Service

# Table of Contents

[Table of Contents 1](#_Toc162910086)

[Introduction 3](#_Toc162910087)

[Presentation Layer 4](#_Toc162910088)

[Leveraging Django's Templating System 4](#_Toc162910089)

[Responsive Design with Bootstrap 4](#_Toc162910090)

[Business Logic Layer 5](#_Toc162910091)

[Utilizing Django Views for Application Logic 5](#_Toc162910092)

[Core Functionalities 5](#_Toc162910093)

[Form Handling Logic 6](#_Toc162910094)

[Data Access Layer 6](#_Toc162910095)

[Architectural Foundation 6](#_Toc162910096)

[Model Overview 6](#_Toc162910097)

[Ensuring ACID Properties 8](#_Toc162910098)

[Security Layer 8](#_Toc162910099)

[Introduction to Django's Built-in Security Features 8](#_Toc162910100)

[CSRF Protection: 9](#_Toc162910101)

[Password Hashing: 9](#_Toc162910102)

[Secure Handling of User Sessions: 9](#_Toc162910103)

[Additional Security Measures 9](#_Toc162910104)

[Input Validation: 9](#_Toc162910105)

[Authenticated User Access: 9](#_Toc162910106)

[HTTPS Enforcement: 10](#_Toc162910107)

[Web Services Integration in the Online Payment Platform 10](#_Toc162910108)

[Introduction to Web Services Integration 10](#_Toc162910109)

[Service Selection: 10](#_Toc162910110)

[Implementation Details: 10](#_Toc162910111)

[Example Request: 11](#_Toc162910112)

[Data Format: 11](#_Toc162910113)

[Usage in the Application: 11](#_Toc162910114)

[Security and Error Handling: 11](#_Toc162910115)

[User Manual for Online Payment Platform 12](#_Toc162910116)

[1. Registration 12](#_Toc162910117)

[Screenshot: Registration Page 12](#_Toc162910118)

[2. Logging In 12](#_Toc162910119)

[Screenshot: Login Page 13](#_Toc162910120)

[3. Viewing Account Details 13](#_Toc162910121)

[Screenshot: Account Dashboard 13](#_Toc162910122)

[4. Sending Payments 14](#_Toc162910123)

[Screenshot: Send Payment Page 14](#_Toc162910124)

[5. Viewing Transactions 14](#_Toc162910125)

[Screenshot: Transactions Page 15](#_Toc162910126)

[Conclusion 15](#_Toc162910127)

# Introduction

In the ever-evolving landscape of digital finance, the demand for robust, secure, and user-friendly online payment solutions has skyrocketed. Recognizing this need, our project endeavors to introduce a simplified yet powerful online payment platform, inspired by the functionalities of renowned services like PayPal. Our primary objective is to facilitate seamless monetary transactions between users, thereby democratizing access to digital financial services.

At the heart of our project lies the Django web framework, chosen for its high level of abstraction, built-in security features, and its "batteries-included" approach. Django, a Python-based framework, is renowned for its efficiency in developing complex web applications with minimal code. Python's syntax clarity and Django's robust architecture have significantly accelerated the development process, allowing us to focus on implementing features critical to user experience and security.

The backend of our application, powered by Django, handles user authentication, account management, transaction processing, and security enforcement. Through Django's ORM (Object-Relational Mapping), we effortlessly interact with the database layer, ensuring data integrity and security. The choice of Django also aligns with our goal of future scalability, as its modular design facilitates the integration of additional functionalities as the platform evolves.

For the frontend, we've embraced Bootstrap, a leading open-source toolkit for developing with HTML, CSS, and JS. Bootstrap enables us to craft aesthetically pleasing and responsive designs, ensuring our platform's accessibility across various devices and screen sizes. Its extensive component library has expedited the design process, allowing us to maintain a focus on user experience without compromising on the visual appeal.

Together, Django and Bootstrap form the technological backbone of our project, harmonizing the backend's logical rigor with the frontend's intuitive design. This synergy not only enhances the platform's overall performance and user satisfaction but also underscores our commitment to leveraging cutting-edge technologies to redefine online payment experiences.

As we navigate through the development journey, our vision remains clear - to create a secure, efficient, and user-centric online payment service. A platform that not only meets the current demands of digital transactions but also anticipates future trends, setting a new standard in the realm of online payments.

# Presentation Layer

## Leveraging Django's Templating System

The presentation layer of our online payment platform harnesses the full potential of Django's powerful templating system to deliver a user-friendly interface. Django's templating engine is a cornerstone feature that allows for the dynamic generation of HTML content, making it possible to create rich, interactive user experiences that are both engaging and intuitive.

At the core of Django's templating system is the DRY (Don't Repeat Yourself) principle. Templates can inherit from base templates, ensuring that common elements like headers, footers, and navigation bars are consistent across the platform while reducing code redundancy. This inheritance mechanism is particularly beneficial for maintaining a uniform look and feel throughout the application, enhancing the user's navigation experience.

We have meticulously designed custom templates for key functionalities such as user registration, login, account management, transaction listing, and payment initiation. Each template is crafted to not only meet the specific needs of the functionality it represents but also to ensure a seamless and harmonious user journey. For instance, the registration and login templates are streamlined to remove any unnecessary complexity, guiding users smoothly through the process with clear instructions and feedback.

The account management template provides users with a clear overview of their account details, including balance and transaction history, presented in a clean, easy-to-understand format. The transaction listing template goes a step further, organizing transaction data in a tabular format that is both easy to scan and interact with, facilitating a better understanding of one's financial activities.

## Responsive Design with Bootstrap

In today's digital age, where users access web applications from a myriad of devices with varying screen sizes, responsive design is not just a luxury but a necessity. To address this need, we integrated Bootstrap, a leading framework for developing responsive and mobile-first websites.

Bootstrap's grid system and responsive utility classes allow our templates to automatically adjust and optimize layout based on the device's screen size. This ensures that whether a user accesses our platform from a desktop, tablet, or smartphone, the experience remains consistent and fully functional. Bootstrap's component library has also been instrumental in implementing features such as modals for confirming transactions, alerts for providing immediate feedback, and forms styled for clarity and ease of use.

The integration of Bootstrap into our Django templates has significantly enhanced the aesthetic appeal and usability of our platform. By adhering to Bootstrap's design principles, we ensure that the user interface is not only visually appealing but also accessible and user-friendly across all device sizes. The consistency in design elements, such as buttons, forms, and navigation menus, fosters a sense of familiarity and confidence among users, encouraging longer engagement and repeat usage.

The presentation layer of our online payment platform represents a fusion of Django's templating capabilities with Bootstrap's responsive design framework, culminating in a user interface that is both beautiful and functional. Through the thoughtful design of custom templates, we have created a user experience that simplifies digital transactions, making them more accessible and enjoyable. As we continue to evolve and expand our platform, the principles of user-centric design and responsive layout will remain at the forefront of our development strategy, ensuring that we meet and exceed the expectations of our users in an ever-changing digital landscape.

# Business Logic Layer

## Utilizing Django Views for Application Logic

The business logic layer of our online payment platform is intricately designed using Django's views, which serve as a crucial bridge between the presentation layer (templates) and the data access layer (models). Django offers two types of views – class-based views (CBVs) and function-based views (FBVs) – each providing a structured method to handle the logic behind user requests.

**Class-Based Views (CBVs):** Our platform heavily utilizes CBVs for their modularity and reusability. CBVs are particularly advantageous for standard CRUD (Create, Read, Update, Delete) operations. For instance, the **SignUpView** extends Django's generic **CreateView** to handle user registration. Similarly, **ListView** and **DetailView** are employed to list transactions and display account details, respectively. CBVs allow for clear separation of concerns by encapsulating specific functionalities within classes, making the codebase more organized and easier to maintain.

**Function-Based Views (FBVs):** While CBVs handle the bulk of standardized operations, FBVs are used for more customized or complex logic that requires a procedural approach. For example, the process of initiating a payment involves several steps – fetching user account details, validating transaction data, updating account balances – which are succinctly managed within an FBV due to its straightforward, linear execution flow.

## Core Functionalities

**User Registration and Authentication:** The **SignUpView** not only creates a new user record using Django's built-in **UserCreationForm** but also extends this functionality to include currency selection for the user's account. Upon successful form submission and user creation, an **Account** instance with a zero balance and selected currency is created, linking it to the newly registered user. Django's authentication system is then leveraged to manage user login sessions, ensuring that users can securely access their accounts.

**Account Balance Management:** Account balance management is central to our platform, enabling users to view their current balance and execute transactions. Django views interact with the **Account** model to fetch and display balance information. When a transaction is initiated, the platform employs transactional integrity (using Django's **@transaction.atomic** decorator) to update the sender's and receiver's account balances atomically, safeguarding against data inconsistencies.

**Transaction Processing:** The transaction initiation process is meticulously designed to ensure accuracy and security. Users can send payments to others by specifying the receiver's details and the amount. The system first verifies the availability of funds and the existence of the receiver's account before proceeding to update the accounts' balances. A **Transaction** record is then created to log the transaction, providing an immutable history of all user transactions.

## Form Handling Logic

Form handling in our platform is integral to data collection and validation. Django forms not only simplify the rendering of HTML form elements but also enforce validation rules, ensuring that submitted data is accurate and secure. Our custom **UserCreationForm** includes an additional field for currency selection, demonstrating how Django forms can be customized to meet specific requirements. This form is intricately linked with the user registration logic, illustrating the seamless integration between Django's form handling capabilities and the application's business logic.

The form's **clean** methods are overridden when necessary to implement custom validation logic, such as ensuring unique usernames or validating transaction amounts against account balances. This robust form handling mechanism significantly enhances the platform's security and user experience by preventing erroneous or fraudulent data submissions.

The business logic layer of our online payment platform is a testament to the flexibility and power of Django's views and form handling mechanisms. Through the judicious use of class-based and function-based views, coupled with comprehensive form validation, we've constructed a robust backbone for managing user interactions, authentication, account management, and transaction processing. This layer not only ensures the smooth operation of our platform but also lays a solid foundation for scaling and introducing new features in the future, all while maintaining a focus on security and user experience.

# Data Access Layer

## Architectural Foundation

At the core of our online payment platform lies the data access layer, a critical component engineered to facilitate efficient and secure interactions with the database. This layer is built upon Django's Object-Relational Mapping (ORM) system, which abstracts the complexities of database operations into Python objects, enabling seamless data manipulation and query operations. The models defined in this layer — **User**, **Account**, and **Transaction** — form the backbone of our application, each serving a distinct purpose in the overall system architecture.

## Model Overview

**User Model:** Django's built-in **User** model serves as the primary entity for authentication and user management. It encapsulates essential attributes such as username, password, email, etc., and provides a robust authentication framework. Our platform extends this functionality to accommodate additional user-related data.

**Account Model:** The **Account** model is a custom model designed to represent a user's financial account within the platform. It is defined as follows:

from django.db import models

from django.conf import settings

class Account(models.Model):

    user = models.OneToOneField(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

    balance = models.DecimalField(max\_digits=10, decimal\_places=2)

    currency = models.CharField(max\_length=3, choices=(('GBP', 'British Pound'), ('USD', 'US Dollar'), ('EUR', 'Euro')))

    def \_\_str\_\_(self):

        return f"{self.user.username}'s account"

This model establishes a one-to-one relationship with the **User** model, ensuring each user has a unique account. It tracks the account balance and currency, pivotal for executing and recording transactions.

**Transaction Model:** The **Transaction** model captures the details of financial transactions between users:

class Transaction(models.Model):

    sender = models.ForeignKey(settings.AUTH\_USER\_MODEL, related\_name='sent\_transactions', on\_delete=models.CASCADE)

    receiver = models.ForeignKey(settings.AUTH\_USER\_MODEL, related\_name='received\_transactions', on\_delete=models.CASCADE)

    amount = models.DecimalField(max\_digits=10, decimal\_places=2)

    currency = models.CharField(max\_length=3, choices=(('GBP', 'British Pound'), ('USD', 'US Dollar'), ('EUR', 'Euro')))

    timestamp = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"Transaction from {self.sender.username} to {self.receiver.username}"

This model uses foreign keys to link to the **User** model for both the sender and receiver, allowing for the tracking of sent and received transactions. The **amount**, **currency**, and **timestamp** fields record the transaction details.

Leveraging Django ORM for Data Manipulation and Query Operations

Django ORM acts as an intermediary between the Python code and the database, allowing for database operations without writing raw SQL. This abstraction enables developers to work within the Python codebase, maintaining readability and leveraging Python's syntax.

For instance, to update an account's balance after a transaction, the ORM facilitates a straightforward operation like:

account.balance += transaction.amount

account.save()

Similarly, querying the database for a user's transactions becomes intuitive:

user.sent\_transactions.all()

## Ensuring ACID Properties

Django's ORM is built to ensure that transactions adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, crucial for the integrity of financial transactions. The ORM, combined with the underlying database's transaction management features, ensures that operations are atomic, maintaining data consistency even in the event of a system failure or concurrent access scenarios.

The **@transaction.atomic** decorator is extensively used to wrap sequences of operations that must be executed as a single unit. This ensures that either all operations succeed, or none do, thereby preserving the integrity of financial transactions and account balances within the platform.

The data access layer, underpinned by Django's ORM, provides a robust foundation for our online payment platform. It simplifies complex database interactions, ensuring data integrity and security while enabling scalability. The thoughtful design of the **User**, **Account**, and **Transaction** models, along with the strategic use of ORM features to ensure ACID properties, sets the stage for a reliable and efficient financial transaction system. This layer not only facilitates the core functionalities of our platform but also ensures that user data is handled securely and responsibly, fostering trust and reliability in the digital payment ecosystem.

# Security Layer

## Introduction to Django's Built-in Security Features

The security layer of our online payment platform is fortified using Django's comprehensive suite of built-in security features. These features are designed to protect against a wide array of vulnerabilities, ensuring the integrity, confidentiality, and availability of user data and transactions. By leveraging Django's security mechanisms, our platform establishes a robust defense against common web application threats.

## CSRF Protection:

Cross-Site Request Forgery (CSRF) attacks trick a user into executing unintended actions on a web application where they're authenticated. Django mitigates this risk with CSRF tokens, a unique, random token that is checked on every state-changing POST request. The CSRF middleware automatically includes a token in forms and verifies the token upon form submission, thus ensuring that requests originate from the application's forms. This feature is crucial for our platform, particularly in forms related to transactions and account settings, providing an essential layer of security against unauthorized actions.

## Password Hashing:

Django employs password hashing to secure user passwords. When a user creates or updates their password, Django converts it into a hash—a fixed-size string of characters generated from the input password—using the PBKDF2 algorithm with a SHA256 hash, by default. This hashed version is stored in the database, ensuring that actual passwords are never saved or transmitted in clear text. In the event of unauthorized database access, attackers cannot easily decipher user passwords, significantly reducing the risk of compromised accounts.

## Secure Handling of User Sessions:

Django's session framework manages user sessions, allowing users to log in and out of the platform securely. Sessions are stored server-side, and only an encrypted session key is stored on the client's browser as a cookie. This key is used to retrieve the session data, keeping user-related data secure. The framework also provides session expiration, ensuring that old or idle sessions are invalidated, further protecting user accounts from unauthorized access.

## Additional Security Measures

Beyond Django's built-in features, our platform implements several other security measures to safeguard against vulnerabilities and enhance data protection.

### Input Validation:

Input validation is crucial to prevent injection attacks, such as SQL Injection or Cross-Site Scripting (XSS). Django forms inherently validate input data against the expected data types and constraints. Custom validators are also implemented to ensure complex inputs, like transaction amounts or new account details, adhere to our specific rules, such as minimum and maximum values, to prevent abnormal or malicious data submissions.

### Authenticated User Access:

Access control is a fundamental aspect of our security strategy. Django's authentication and permissions system is utilized to restrict access to certain views and operations based on the user's authentication status and roles. Decorators like **@login\_required** and mixins such as **LoginRequiredMixin** or **UserPassesTestMixin** are employed extensively to protect views that manage sensitive operations, ensuring that only authenticated and authorized users can execute specific actions, such as initiating transactions or accessing account details.

## HTTPS Enforcement:

To protect data in transit, our platform enforces HTTPS, ensuring that all data transmitted between the client and server is encrypted. Django's **SecurityMiddleware** is configured to redirect all HTTP requests to HTTPS, and the **SECURE\_SSL\_REDIRECT** setting is enabled to ensure this behavior is consistent across the platform. This encryption prevents attackers from eavesdropping on or tampering with data, safeguarding user credentials and transaction details.

The security layer of our online payment platform represents a comprehensive approach to safeguarding user data and transactions. By effectively leveraging Django's built-in security features and implementing additional security measures, we establish a secure environment that users can trust. From CSRF protection and password hashing to secure session management and beyond, our commitment to security is evident in every aspect of the platform's design and operation. As we continue to evolve and enhance our platform, maintaining the highest standards of security will remain a top priority, ensuring the protection and privacy of our users in the digital payment landscape.

# Web Services Integration in the Online Payment Platform

## Introduction to Web Services Integration

In the modern era of interconnected applications, web services play a pivotal role in extending the functionality of web-based platforms by enabling them to interact with external services and data sources. Our online payment platform leverages web services, specifically for currency conversion, to provide real-time financial data and ensure that transactions across different currencies are processed accurately.

Integrating Currency Conversion Web Services

### Service Selection:

For currency conversion, our platform integrates with a third-party web service that offers up-to-date exchange rates for a wide range of currencies. This service is selected based on its reliability, ease of integration, and the accuracy of its financial data.

### Implementation Details:

The integration is implemented using Django's **requests** library, which simplifies HTTP communications. Our application primarily uses the HTTP GET method to retrieve currency conversion rates, as this operation does not modify any data on the web service and merely requests information.

### Example Request:

import requests

def get\_conversion\_rate(from\_currency, to\_currency):

    url = f"https://api.exchangeratesapi.io/latest?base={from\_currency}&symbols={to\_currency}"

    response = requests.get(url)

    data = response.json()

    return data['rates'][to\_currency]

In this example, the function **get\_conversion\_rate** constructs a URL to query the web service for the exchange rate between two specified currencies. It sends a GET request to the web service and parses the JSON response to extract the conversion rate.

### Data Format:

The communication with the currency conversion web service utilizes JSON (JavaScript Object Notation), a lightweight data-interchange format. JSON is chosen for its human-readable format and its compatibility with most programming languages, making it an ideal choice for web service communication. The JSON response typically contains keys representing different currencies and their corresponding rates relative to the base currency specified in the request.

### Usage in the Application:

Within our platform, the currency conversion functionality is utilized in several scenarios, such as when displaying account balances in a user's preferred currency or processing transactions involving different currencies. By integrating this web service, the platform can dynamically convert amounts based on real-time exchange rates, enhancing the user experience by providing accurate financial information.

### Security and Error Handling:

To ensure the security of web service communications, our platform uses HTTPS for all requests, encrypting data in transit. Additionally, robust error handling mechanisms are in place to address scenarios where the web service is unavailable or returns an error. These measures prevent the application from presenting incorrect data to users and allow for graceful handling of any issues encountered during web service interactions.

The integration of web services for currency conversion represents a critical enhancement to our online payment platform, enabling it to support a global user base with diverse currency needs. By leveraging Django's capabilities for web service communication and adhering to best practices for security and error handling, our platform ensures that users have access to accurate, real-time financial data, making transactions more transparent and trustworthy.

# User Manual for Online Payment Platform

Welcome to our Online Payment Platform User Manual. This guide will walk you through the primary functionalities of our platform, including registration, logging in, viewing account details, sending payments, and viewing transactions.

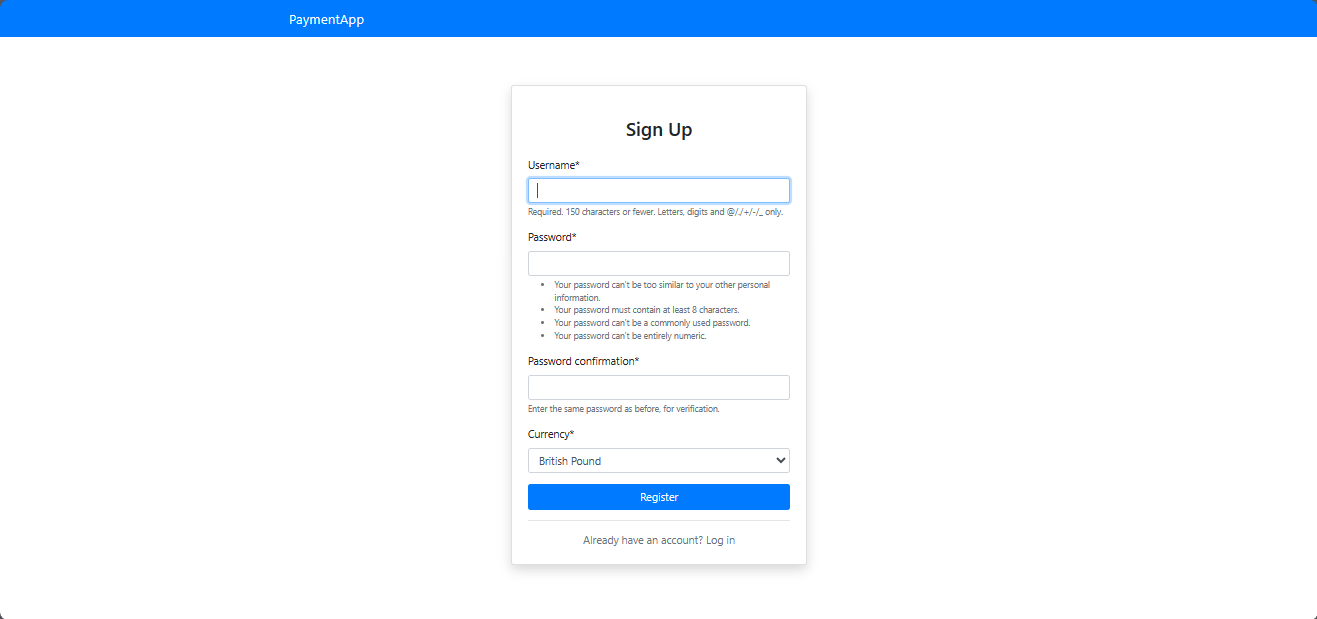
## Registration

**Step 1:** Visit the homepage of the Online Payment Platform. On the top navigation bar, click on the “Sign Up” button.

**Step 2:** You will be directed to the registration page. Fill in the required fields: username, email, password, confirm password, and select your preferred currency from the dropdown menu.

**Step 3:** After filling in the details, click on the “Register” button to create your account. You will receive a confirmation message on the screen once your account is successfully created.

### Screenshot: Registration Page



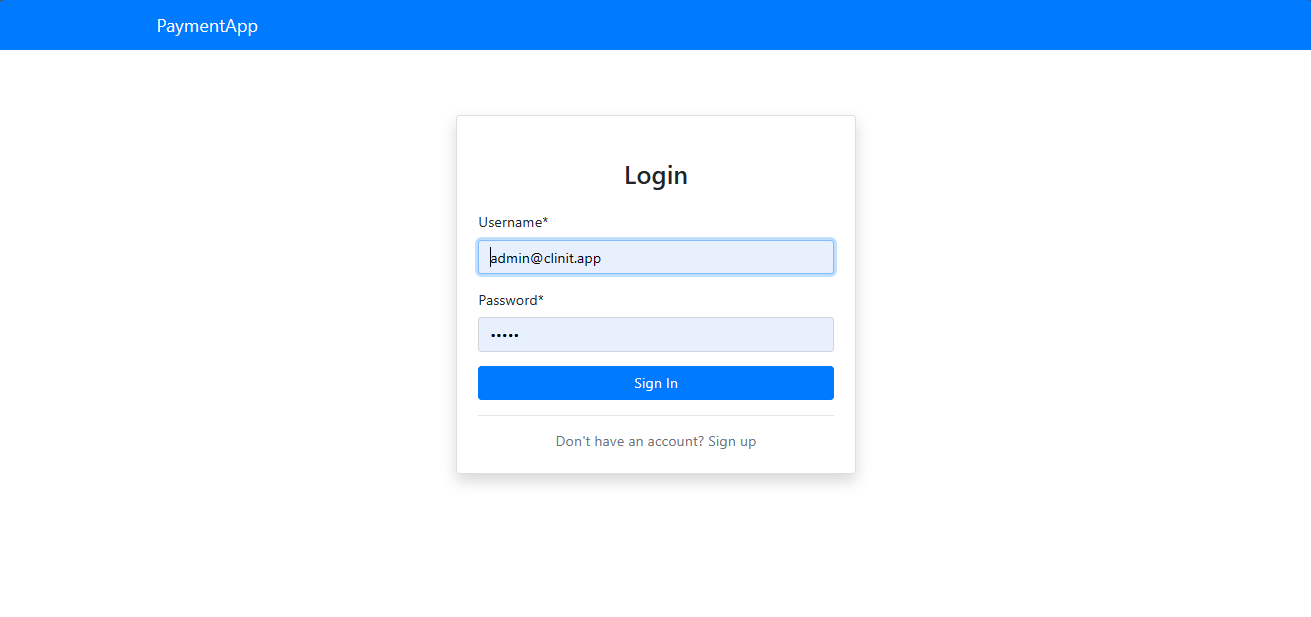
## Logging In

**Step 1:** From the homepage, click on the “Login” button located in the top navigation bar.

**Step 2:** Enter your username and password in the respective fields on the login page.

**Step 3:** Click the “Login” button to access your account. If your credentials are correct, you will be redirected to your account dashboard.

### Screenshot: Login Page

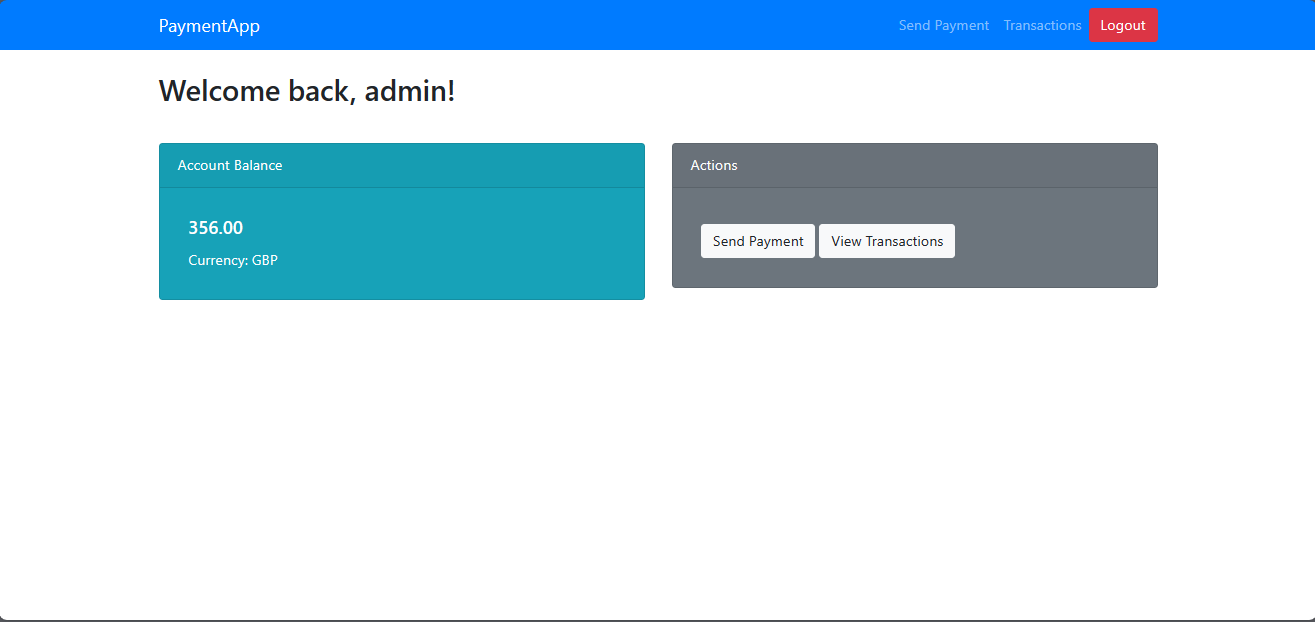


## Viewing Account Details

**Step 1:** Once logged in, you are directed to the dashboard. Here, you can view your account balance and currency displayed prominently.

**Step 2:** For more details, click on the “Account Details” button. This page will show your account balance, currency, and other relevant account information.

### Screenshot: Account Dashboard



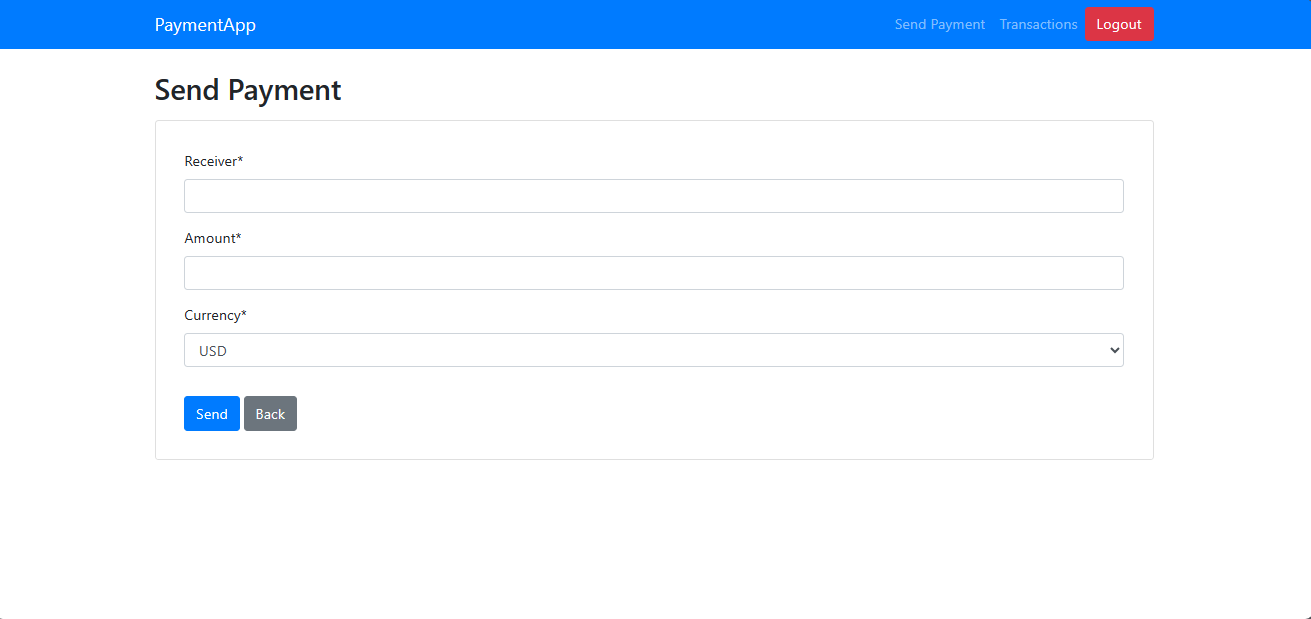
## Sending Payments

**Step 1:** To send a payment, navigate to the “Send Payment” button on the dashboard or in the top navigation menu.

**Step 2:** On the Send Payment page, fill in the details of the transaction: the recipient’s username, amount, and currency.

**Step 3:** Review the information, then click “Send” to complete the transaction. A confirmation message will appear once the payment is successfully processed.

### Screenshot: Send Payment Page



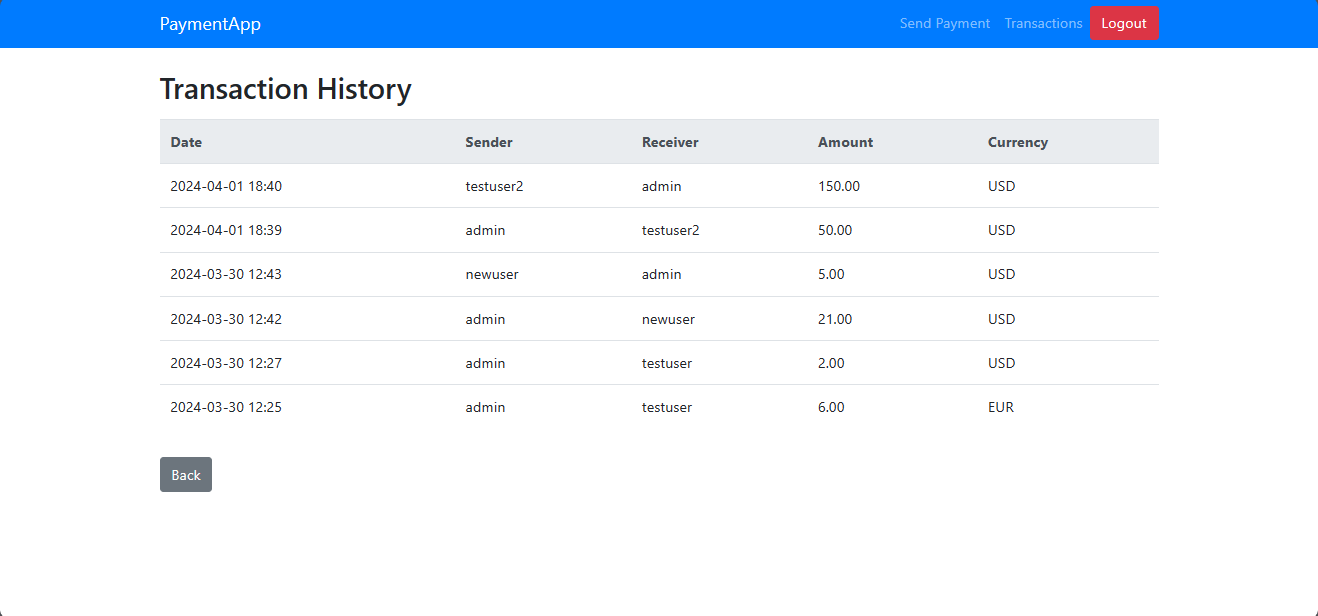
## Viewing Transactions

**Step 1:** To view your transaction history, select the “View Transactions” option from the dashboard or the top navigation menu.

**Step 2:** You’ll be taken to the Transactions page, where a list of all your transactions, including sent and received payments, are displayed along with details such as date, sender, receiver, amount, and currency.

**Step 3:** You can click on any transaction to view more detailed information if this functionality is supported.

### Screenshot: Transactions Page



**Tips for a Smooth Experience**

* Keep your login credentials secure.
* Regularly review your transaction history for any unauthorized activity.
* Use the logout button to securely exit your account after each session.

This manual provides a foundation for getting started with our Online Payment Platform. For further assistance or more detailed inquiries, please refer to our FAQ section or contact our support team.

# Conclusion

The completion of this online payment platform project marks a significant milestone in our journey to simplify and secure digital transactions for users worldwide. Through the careful integration of Django's robust backend capabilities with Bootstrap's responsive frontend design, we have crafted an application that not only meets the technical demands of online payment processing but also delivers an intuitive and seamless user experience.

Our exploration into Django's versatile features, from its templating system and ORM to its comprehensive security framework, has allowed us to construct a solid foundation that supports the core functionalities of registration, login, account management, transaction processing, and viewing transaction history. The utilization of class-based and function-based views has enabled us to implement complex business logic efficiently, while the custom form handling and user authentication mechanisms ensure that user interactions are both secure and user-friendly.

The integration of web services for currency conversion highlights our commitment to providing real-time, accurate financial data, a critical component in the facilitation of cross-currency transactions. This feature, coupled with Django's ORM capabilities, ensures that our platform operates with the highest levels of data integrity and consistency, adhering to ACID properties and protecting against common web vulnerabilities.

Security has been a paramount concern throughout the development process. By leveraging Django's built-in security features, such as CSRF protection, password hashing, and secure session management, in conjunction with additional measures like input validation and HTTPS enforcement, we have established a secure environment that users can trust. This dedication to security not only safeguards user data but also enhances the overall reliability of the platform.

The creation of a detailed user manual, complete with step-by-step guides and annotated screenshots, underscores our commitment to accessibility and user support. This resource is designed to empower users, enabling them to navigate the platform confidently and make the most of its features.

In conclusion, this project represents not just a technical achievement, but a forward step in our mission to enhance the digital payments landscape. The knowledge gained, challenges overcome, and the feedback received throughout this project will serve as invaluable assets as we continue to refine and expand our platform. Looking ahead, we are excited about the possibilities that lie in further innovation, integration of new technologies, and the expansion of our services to meet the evolving needs of our users in an increasingly digital world.